```
template <int DDL, int MDL, int TDL, int DDR, int MDR, int TDR>
auto operator/(Unit<DDL,MDL,TDL> lhs, Unit<DDR,MDR,TDR> rhs)
{
    return Unit<DDL-DDR, MDL-MDR, TDL-TDR>(lhs.value() / rhs.value());
}
```

The return types for the multiplicative operators are somewhat awkwardly long, and without deduced return types, those long names would need to appear twice, once in the function declaration and once in the **return** statement.

As a workaround, **operator*** and **operator/** could introduce a defaulted **template parameter** to avoid the repetition of the return type:

```
template <int DD1, int MD1, int TD1, int DD2, int MD2, int TD2,
        typename R = Unit<DD1+DD2, MD1+MD2, TD1+TD2>>
R operator*(Unit<DD1,MD1,TD1> lhs, Unit<DD2,MD2,TD2> rhs)
{
    return R(lhs.value() * rhs.value());
}
```

However, the workaround does not apply to ~~nontemplated functions~~, such as `kineticEnergy`.

We can now use these operations to implement a function that returns the kinetic energy of a moving object:

```
auto kineticEnergy(Kilograms m, Mps v)
    // Return the kinetic energy of an object of mass m moving at velocity v.
{
    return m * (v * v) / Scalar(2);
}
```

The return type of this formula is determined automatically, without expressing the `Unit` **template arguments** directly. The returned unit is a joule, which can also be described as a kilogram $*$ meter$^2$/second$^2$, as our test program illustrates:

```
#include <type_traits>  // std::is_same

void f1()
{
    using Joules = Unit<2, 1, -2>;  // energy in joules

    auto ke = kineticEnergy(Kilograms(4.0), Mps(12.5));
    static_assert(std::is_same<decltype(ke), Joules>::value, "");
}
```

Because of automatic return-type deduction, naming the `Unit` instantiation of each intermediate computation within `kineticEnergy` was unnecessary. The **static_assert** in the code above proves that our formula has returned the correct final unit.